**Final Year Project**

## **Abstract**

* Objective: research ML/AI approaches to predict price of a stock, develop several models, compare and analysis result
* Method summary
* Findings
* Summary

Contents

[**1.** **Abstract** 1](#_Toc195872529)

[**2.** **Introduction** 2](#_Toc195872530)

[**3.** **Literature Review** 2](#_Toc195872531)

[3.1. Introduction 2](#_Toc195872532)

[3.2. Efficient Market Hypothesis 2](#_Toc195872533)

[3.3. Statistical models 2](#_Toc195872534)

[3.5. Artificial Neural Networks (ANN) 4](#_Toc195872535)

[**4.** **Methodology** 5](#_Toc195872536)

[4.1. General Method 5](#_Toc195872537)

[4.2 Pre - ML/AI methods. 9](#_Toc195872538)

[4.3 Machine Learning Techniques 11](#_Toc195872539)

[4.4 Artificial Neural Networks (ANNs) 13](#_Toc195872540)

[5.5 Further predictions 14](#_Toc195872541)

[**5.** **Results and Data visualisation** 14](#_Toc195872542)

[5.1. Pre-ML/AI methods Results. 14](#_Toc195872543)

[5.2 Machine Learning Techniques Results 16](#_Toc195872544)

[5.4 Artificial Neural Networks 22](#_Toc195872545)

[5.5 Further predictions 24](#_Toc195872546)

[**6.** **Comparison and discussion** 24](#_Toc195872547)

[**7.** **Discussion** 25](#_Toc195872548)

[**8.** **Conclusion** 25](#_Toc195872549)

[**9.** **References** 25](#_Toc195872550)

## **Introduction**

## **Literature Review**

### 3.1. Introduction

Due to the potential for financial reward, predicting the changes in the price of a stock is a popular area of research. Models for these predictions have increased in number and complexity over time; from simple models such as moving averages, to early applications of machine learning methods, to complex artificial neural networks. This literature review focuses on the technical side of prediction, using historical data to predict future results. It attempts to highlight the performance of these models and techniques, creating a benchmark for comparison for the models I create.

### 3.2. Efficient Market Hypothesis

Before looking at the models, it is essential to address whether stock markets can be predicted. Fama (1965) states one hypothesis is the efficient market hypothesis (EHM), which theorises that the price of the stock reflects all available information about it and therefore, forecasting future prices is impossible This hypothesis supports the random walk hypothesis, which suggests that movement in a stock’s price is random, dictated by emerging and unpredictable information.

However, there is significant critique to these hypotheses, supporting a semi-strong form of the efficient market hypothesis, that only publicly available data is reflected in the price of a stock (Jensen, M. C.,1978). As such you cannot consistently predict the market with publicly available data, you would need insider knowledge. However, Jensen, M. C’s article gives evidence that there are anomalies suggesting that there is a flaw to the semi-strong EHM.

If the markets are not fully efficient, then by analysing publicly available data, which may not be fully reflected in the price of a stock, you can achieve a prediction outperforming random chance, leading to financial gain.

### 3.3. Statistical models

The most naïve model for predicting the stock, supported by the efficient market hypothesis is to simply use the latest price as the prediction.

A slightly more complex approach involves simple moving averages (SMA). A SMA is the average price of a stock over a given number of days. These are used as indicators of the trend of a stock. If a stock’s current price is above the moving average, it is said to be on an upwards trend. Similarly, if the current price is below the moving average, it is said to be on a downwards trend. While moving averages are a good indicator for the trend of a price, due to their nature, they lag behind the actual price movement, so may not be sufficient for short-term predictions.

A more advanced statistical model is the autoregressive integrated moving average (ARIMA). ARIMA models have been used in a range of industries for time series analysis. Such examples include Irish Inflation (Meyler, A., Kenny, G., and Quinn, T, 1998).

Dell Stock Prediction  
Adebiyi, Adewumi, and Ayo (2014) applied ARIMA to predict Dell's stock prices from August 1988 to February 2011. Their model achieved:

* Bayesian Information Criterion (BIC): 4.5588.
* Adjusted R2R^2R2: 0.9897.
* Standard Error of Regression (SER): 2.361.

NYSE and Nigerian Stock Exchange Predictions  
Adebiyi, A. A., Adewumi, A. O. and Ayo, C. K (2014) used ARIMA to predict stocks on the New York Stock Exchange and Nigerian Stock Exchange. Their model achieved the metrics of:

* BIC: 23,736
* R2: 0.9972
* SER: 0.7872

In 2018, Almasarweh, M. and Al Wadi, S used ARIMA to predict banking stock market data. They used the metric of root mean squared error to compare their models instead and achieved the lowest value of 1.4.  
  
These studies demonstrate the effectiveness ARIMA can have on time series data and by extension stock market predictions. However, it is limited by its linear reliance, failing to map more complex relationships. This is an area machine learning and artificial intelligence models aim to fill.  
  
3.4. Machine Learning Models

Random Forrest

Sadorsky demonstrates the use of random forest to predict the direction of clean energy stocks with an 80% accuracy. They identified that the most important technical indicator was the 200-day moving average and the on-balance volume (OBV)

Support Vector Machine

Meesad, P. and Rasel, R. I. (2013) used support vector regression models to predict the ACI group of company Limited on the Dhaka stock exchange. The used a mean average percentage error (MAPE) to determine their best models. They achieved a MAPE of 0.04 for ‘1 day ahead’ predictions, 0.15 for ‘5 days ahead’ and 0.22 for ’22 days ahead’.

Gradient boosting: TODO

### 3.5. Artificial Neural Networks (ANN)

Adebiyi, A. A., Adewumi, A. O. and Ayo, C. K (2014) compared their ARIMA model to artificial neural network model, they found that the best model for predicting the price of Dell’s stock was three layers in size and consisted of a 10-17-1 architecture. They used the mean squared error and achieved and error of 0.071589 after 5000 epochs

Zhang, G. P. (2003) created a hybrid ARIMA and ANN model, they trained it on sunspots, Canadian lynx data as well as the GPB/USD exchange rate. They used mean squared error and mean absolute deviation to analysis their model over a period of 1, 6 and 12 months, their findings can be seen below.A close-up of a paper

Description automatically generated

## **Methodology**

### 4.1. General Method

#### 4.1.1 Data Collection

Historical stock prices can be sourced from existing libraries such as ‘Yahoo Finance’. A simple API call specifying the label of the stock along with the window of time, will return a data frame containing the date and such values as the ‘Highest Price’ (High), ‘Lowest Price’ (Low), ‘Opening Price’ (Open), ‘Closing Price’(Close) and ‘Volume’ or number shares traded. The models below will attempt to predict the Close but may use the other data to create more indicators or features for it to use. I have decided to use the data that ranges from 01/01/2014 up to 01/01/2023 for testing and training data. I will then use the dates that range from 01/01/2024 till 01/06/2024 to see how well the models extrapolates. Table 1 shows the first 5 dates gathered from yahoo finance alongside the target.

#### 4.1.2 Target Generation

The goal of each model is to create a regression function that takes in the features expressed below and return the difference between the current close and the next day’s close price. I have chosen to do the difference because it abstracts away the actual price of the stock at any given time and so should generalise better and help avoid overfitting. Table 1 shows the first five days’ targets.

A table with numbers and letters

AI-generated content may be incorrect.

Table : First 5 rows gather from yahoo finance

#### 4.1.3 Feature Sanitisation and Transformation

Before we train the models with the data, we need to scale it to reduce any bias towards data with a larger average. The Library Scikit Learn (sklearn 2025) has a Standard scaler and MinMaxScaler the applies a function to each value. The Standard Scaler uses the equation 1 where *x* is the value to be scaled, *u* is the mean of the data and *s* the standard deviation. The MinMaxScaler applies the equations 2 and 3 to each data point, equation 2 scales down the data with respect to the largest and smallest value in the data. Equation 3 is used to scale the data between a given range (*max* and *min)*. Here I have used 1 and -1. Either Scaler can be used, and I’ve used both depending on which scaling produced better results from each model.

Equation : Standard Scaling Equation using Mean and Standard Deviation

Equation First part of min-max scaler

Equation : Second Part of Min-Max Scaler, Determines Scale of the Output

#### 4.1.4 Feature Extraction

To make my models more complex, several different features will be generated using existing data, also known as feature extraction.

##### 4.1.4.1 Lagged Differences

As this is a time series problem. I have added the difference between the close of a given day and the close of a given number of previous days denoted as ‘lagged differences.

##### 4.1.4.2 Simple Moving Average (SMP)

As mentioned in section 3.3, ‘Simple Moving Averages’ takes the mean value of the closes of a given number of days. Having several different sized windows, for example the last 5, 10, 50, 100 days can give a little more insight into how the stock has performed previously, and how the current value compares. I have given the model the simple moving average of the last 10, 50 and the number of given lagged differences.

##### 4.1.4.3 Relative Strength Index (RSI)

The Relative Strength Index (RSI), given by equation 4 is a momentum oscillator that focuses on the speed and strength of a price’s movement. It returns a value between 0 and 100, high values (70 and above) indicate the stock is overbought; that the price has risen quickly and could start decreasing. Similarly, low values (30 and below) indicate the stock is oversold, undervalued and could start increasing.

In Equation 4, The two variables Nup and Ndown represent the average of a given number, n days where the close has gone up and down respectively. For my models I have selected to use the past 14 days

Equation 4: Relative Strength Index Equation

##### 4.1.4.4 Average True Range (ATR)

Given by Equation 5, the Average True Range is another indicator, but measures market volatility. A high ATR signifies that the stock’s price fluctuates a lot, while a low ATR signifies limited change. Similarly, to RSI, it has a variable number of days, again I have chosen 14 days.

Equation 5: Equation for Average True Range

##### 4.1.4.5 Feature Snapshot

The data passed into the model will include, the High, Low, Close, Volume, simple moving average of the past 10, 50 and lagged number of days, the RSI and ATR, the day of week as an integer and the differences of up to a given number of previous closes.

Table 2 shows the first 5 dates of features that will be inputted to each model, before being scaled down. It uses 5 lagged days and therefore has a simple moving average of 5, 10 and 50. This data start on the 14th of march instead of the 1st of January. This is because the first 49 days (not including weekends) cannot create a 50-day simple moving average, therefore I have omitted those dates from the training and testing data.

A close up of a white background

AI-generated content may be incorrect.

Table Features before scaling

#### 4.1.4 Hyperparameter Definitions

Each model has a given number of hyperparameters. I have created several different models of the same type, but with different values for their hyperparameters. I have compared the performances of the models using interpolated data, which produced a set of performance metrics, explained in section 4.1.6 and 4.1.7. Once the best model for each type of machine learning technique was identified, I then compared across model’s types with different, extrapolate data and found the best performing model.

Hyperparameters can be split into two groups.

Model hyperparameters define the structure of the model, the functions and the algorithms that it performs on the data. For example, the number of hidden nodes in an Artificial Neural Network (ANN) or the number of lagged differences that the model expects.

Training/Optimisation hyperparameters affect the training of the model. They can specify how or when to update the weights, when to stop training and how quickly the model changes during each step of training process.

##### 4.1.5 Model Training and Testing

To Train and test the model I separated the data shown above into a training and a testing set. The dates in either set were distributed randomly and split to have 80% of the data in the training set and 20% in the testing set. Over the 10 years this gave us 1772 samples for training and 444 days for testing.

The models were initiated with random weights and their given model hyperparameters and then trained on the training data until it has reached a given condition or has exhausted it. After that the model was then given the testing data at which point it produces a set of predictions.

#### 4.1.6 Compare Metrics

These prediction values were compared against the target values. There are several different metrics for comparing the models. Most of which use the concept of ‘error’ which is simply the actual or percentage difference between the prediction and target value. For example, if I predict the price will be $100 tomorrow, and it is observed at $99, the error is $1 or 1%. Using percentage compared to absolute value allows us to compare models, whose output’s scale differ. Consider I compare two models, one which predicts the change in price and one that predicts the actual price: today’s price is $100, and tomorrow’s price will be $110, both models predict tomorrow’s price will be $120 (the difference model predicts an increase of $20). Both models have an actual error of $10, but the difference model has a percentage error of %100 and the actual value model has a percentage error of 9.1%.

To calculate a single value that I used to compare the model’s performance over a series of predictions, I considered the averages of the actual and percentage errors. Root-Mean-Squared-Error (RMSE) and Mean-Absolute-Error (MAE) combine the actual errors and avoid cancelling out opposite sign errors by squaring or taking the absolute value of the error respectively. RMSE puts significantly more weight on the larger errors as it is squared.

Similarly to MAE, to Mean-Absolute-Percentage-Error (MAPE) takes the absolute value of the percentage error and calculates the average. I decided to use the RMSE and the MAE to compare the performance of different data sets. I did attempt to record the MAPE but because of the frequency of zero change encountered, the metric became skewed.

### 4.2 Pre - ML/AI methods.

These models include a simple moving average (SMA), exponential smoothing and Autoregressive integrated moving average (ARIMA). The performances of these models are used as a benchmark for the other models to determine how their performances compare. The Simple moving average and exponential smoothing models do not require a training and testing split because they do not have any variable weights to train. As they do not need training, they will only be used to predict the extrapolated data and not the training. Some models require several days lag for example the 10-day simple moving average needs data from the last 10 days. As a result, some of the data is omitted, however there are still enough sample to gain enough of a benchmark.

#### 4.2.1 Simple Moving Average (SMA)

As mentioned in section 3.3 and 4.1.3, Simple Moving Average takes the average close of a given number of days. I have taken the simple moving average of the change in price of the past 1,3,5,10 and 50 days and the next day’s difference as the predicted value. The most trivial model is a simple moving average of one which is simply the current days value, so a difference of 0.

#### 4.2.2 Exponential Smoothing

A possible improvement to SMA is to apply weights to the values of the given number of previous days, giving more weight to more recent values. One way I considered was exponential smoothing. Equation 6 shows the recursive formula for exponential smoothing, where alpha is a given constant, St is the prediction at time t, yt is the observed value at time t, St – 1 is the prediction for the day before. To calculate an exponential smoothing prediction for each day, I created windows of varying size and applied the equation to the window and used the final St as the prediction for yt+ 1. The value of alpha could be loosely considered as the first hyper parameter, of which I used several different values of alpha (0.1, 0.2, 0.5, 0.7 and 1)

Equation 6: recursive formula for exponential smoothing

#### 4.2.3 Autoregressive Integrated Moving Average (ARIMA)

Autoregressive Integrated Moving Average (ARIMA) models can be split into three parts.

1. Auto regressive (AR). Given by equation 6, yt is the value at time t, yt-1, yt-2, … are the previous values and ϕ1, ϕ2, … are coefficients and ϵt is the error. Φ is estimated during the training process of the model but is outside the scope of this text. This part of the model takes a value of **p** which represents the number of previous time steps to consider in the equation.
2. Integrated (I). By differencing the steps, we make the series stationary, that is has a constant mean, variance and the relationship with its previous values are stationary. It removes the trend and any seasonal patterns in the data, such as the general upwards trend we can see in figure 4 and 5. Equation 7 shows us the first-order differencing, which is simply the difference between the steps. Higher order differencing takes the difference between the previous order differencing. The order of differencing is given as a parameter **d**.
3. The Moving Average (MA), shown by Equation 8, uses the mean of the series and the error of the past **q** terms multiplied by given coefficients that are estimated during the training or fitting stage of the process

Equation 9 gives us the combination of the models. The final Arima model uses a combination of the Auto Regressive part, the moving average and a constant on the differenced series. After going through the process given in section 4.1, the results are shown in Figure X and Table X

Equation 6: Auto Regressive Equation

Equation 7: Integrated Equation

Equation 8: Moving Average Equation

Equation 9 ARIMA model equation

### 4.3 Machine Learning Techniques

To implement the methods in this section, I use the scikit learn library. The data is gathered from yahoo finance, then the data is scaled, model built, trained and predictions analysed with Scikit learn. Scikit learn has a grid search method which allows you to train and compare different hyper parameters and compare them. It allows cross validation which means that the training data will be spilt again, and the models will be scored in training with the least error achieved from a given number of random splits.

Before I considered ANNs, I research several different techniques. Both Random Forrest and Gradient Boosting use decision trees, which split the data based on some criteria of a feature. For example, if the Close is above a certain value, then that data goes to one child and another if not. The model then combines the output of each of the trees weighted with a learning rate.

The training portion of this model focuses on finding the best feature to split and with what value to minimize the error of the regression. The Tree continually splits until it reaches maximum depth or achieves a minimal number of samples in the leaf node.

##### 4.3.1 Random Forest

Random Forest is an ensemble method, it creates and trains several different models, in this case decision tress and averages their predictions as a final value. The ensemble method that random forest utilizes is called Bagging or Bootstrap Aggregating. It trains different models on different random subsets of the data (sampling with replacement).

* Number of trees: [50,100, 200]
* Max depth of the tress: [None, 10, 20, 30]
* Minimum samples in a split: [2, 5, 10]
* Minimum samples in a leaf: [1,2,4]
* Days lagged: [1,3,5,8,13,21, 40]

##### 4.3.2 Gradient Boosting

Gradient Boosting is another ensemble method, however unlike random forest, it works sequentially. Each subsequent tree aims to correct the errors of the previous one, minimising the loss function. There are several different hyper parameters so instead of a grid search I used a randomised search, to decrease training time. Each hyperparameter is instead given a distribution and each model is initialised with random hyper parameters.

The Gradient Boosting model had the following hyperparameter distribution, where ‘uniform’ means sampled at random between an upper and lower bound, and ‘random-integer’ meant a random integer between the two values

* Learning rate: uniform (0.01, 0.3)
* Maximum Iterations': random-integer (100, 500)
* Maximum depth': random-integer (3, 10)
* Minimum samples in a leaf: random-integer (10, 50)
* l2\_regularization: uniform (0.0, 1.0)
* Maximum bins: random-integer (128, 256)
* Scoring: ['loss', 'neg\_root\_mean\_squared\_error']

‘L2 regularisation’ is a value applied at the leaf nodes, it is multiple by the output of a specific leaf node and added to the loss or error function.

‘Maximum bins’ is a hyperparameter that only applies when using a histogram based gradient boosting regressor. It maps continuous values into discrete ranges.

##### 4.3.4 K-Nearest-Neighbour (KNN)

K-Nearest-Neighbour is a technique that stores the training data and then when given a testing sample, computes the given number, k, closest neighbours in the training data. It then returns the average of the values of the k closest neighbours, potentially weighted.

The hyperparameters of a KNN model are

* Number of Neighbours': [3, 5, 7, 9],
* Weights: ['uniform', 'distance'],
* Distance Metric: ['Euclidean', 'Manhattan']

The number of neighbours refers to how many neighbours value should the model consider. Weights refer how to scale the value different neighbours, giving closer neighbours more impact on the average. Distance metric refers to how the distance between two nodes are calculate.

##### 4.3.5 Support Vector Machines (SVM)

Support Vector Machines are used for classification problems, they attempt top find a decision boundary, where either side indicates the sample belongs to a specific class. Instead, the SVM attempts to find a function that deviates from the actual value by at most a given target denoted as ε. When fitting line or curve to data, if the predicted value lies closer than ε, it is considered correctly predicted.

[TODO explain how support vector machine finds the function]

### 4.4 Artificial Neural Networks (ANNs)

4.4.1 Single and Multi-layer Neural Networks

Artificial neural networks (ANNs) are a structure of artificial perceptrons. A perceptron is a function given by equation 10. Where x­i  is a numerical input, wi is the weight applied to the appropriate input, b is a bias and f is an activation function that translates the sum of the product of weights and inputs and the bias into an output.

The Activation function adds nonlinearity to the model. Some activation functions, seen in equations 11,12, 13 include the Sigmoid function and Rectified Linear Unit Function (ReLU) And tanh function. These activation functions were tested and compared against each other as part of the hyper parameter tuning part of the training and testing process.

Anns have several components, an input layer, which in this case is the different lagged days, indicators and other data associated with the stock price of each day; a hidden layer(s) of perceptrons that are mapped to each input layer node or the hidden layer before it and an output layer which will output the model’s prediction.

I used the following hyperparameters with the different accompanied values to train different models

* Hidden Layer Size: [32, 64, 96, 128, 160, 192, 224, 256]
* Activation Function: [ReLU, tanh, Sigmoid]
* Learning Rate: [0.001, 0.01, 0.0001]

Equation 10 Function for a perceptron

Equation 1: Sigmoid function

Equation 1: Rectified Linear Unit

Equation 1: tanh equation

#### 4.4.2 Recurrent and Long-Short-Term Memory Neural Networks.

### 5.5 Further predictions

[TO DO BIC, R^2, SER]

## **Results and Data visualisation**

### 5.1. Pre-ML/AI methods Results.

##### 5.1.1 Simple Moving average (SMA)

Figure 4 shows a graph of the different simple moving averages alongside the actual difference of the stock between the 2023 and 2024. Table shows the performance metrics of each moving average over the 10 years. Using the simple moving averages for prediction we can see the best model is the 3 day moving average.

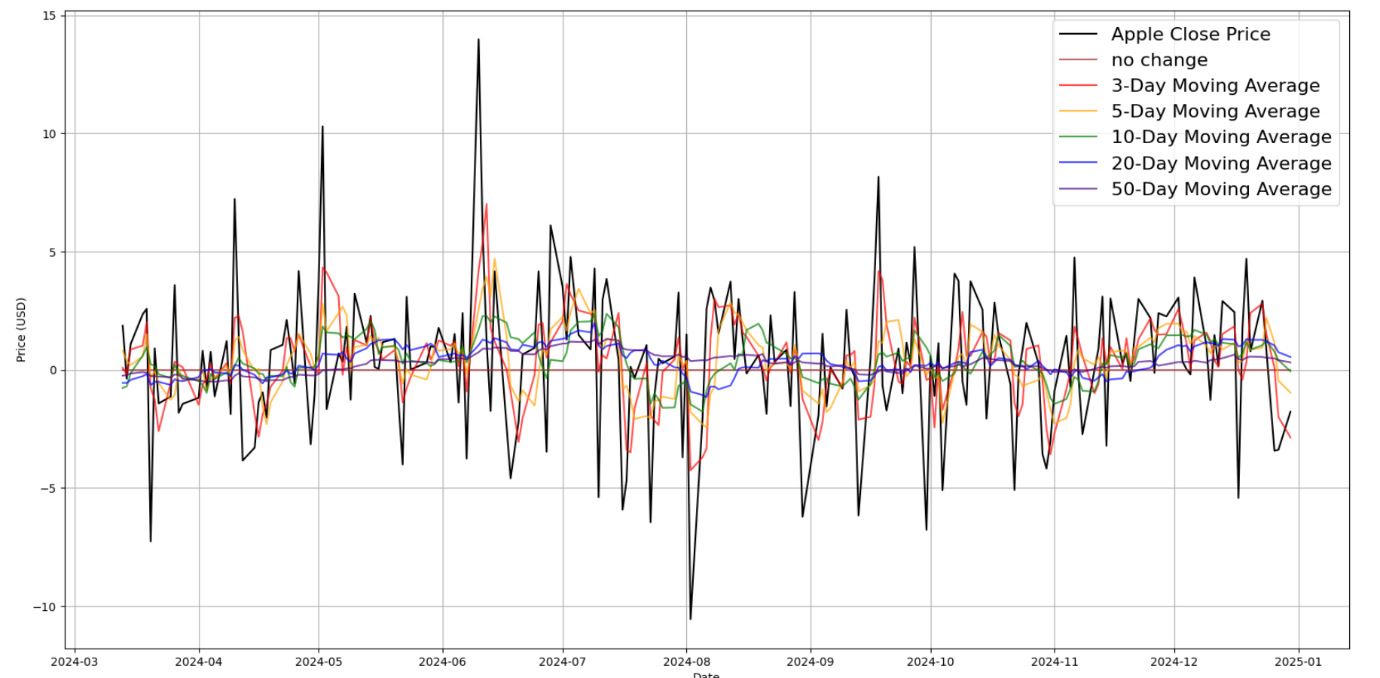


Figure : Graph of simple moving averages of Apple stock between 01/01/2023 and 01/01/2024

|  |  |  |
| --- | --- | --- |
| SMA/Metric | RMSE | RAE |
| No change | 3.035532209045851 | 2.228295656714109 |
| 3 days | 2.499665633732886 | 1.902209769774585 |
| 5 days | 2.6857105397844983 | 2.004021098826191 |
| 10 days | 2.8495726936611114 | 2.0944493964166924 |
| 20 days | 2.94837953147438 | 2.129839411820516 |
| 50 days | 2.988610139197808 | 2.1314892970925507 |

Table 3: Error Metrics of Simple Moving Average

##### 5.1.2 Exponential smoothing

Figure 5 shows the different predictions from each of the models with table 4 shows their performance metrics. The best performing model over all window sizes used an alpha value of 0.1; it also had a slight improvement over the best SMA model.

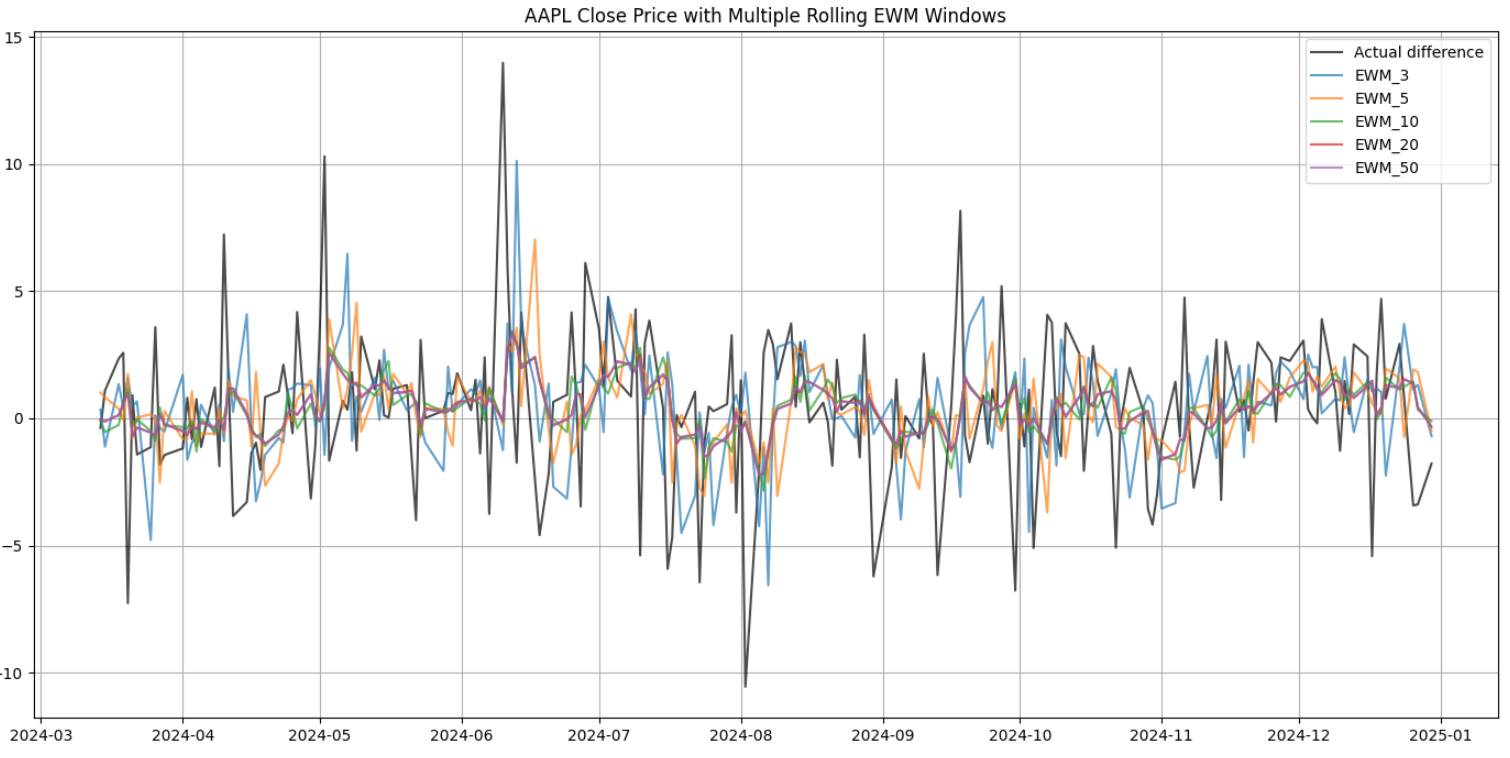


Figure : exponential smoothing with alpha = 0.7

|  |  |  |
| --- | --- | --- |
| Window Size/Metric | RMSE | RAE |
| 3 days | 2.4356847947175737 | 1.8916985439894787 |
| 5 days | 2.3730193480169968 | 1.7971548810949276 |
| 10 days | 2.1625382805680453 | 1.6882754830288162 |
| 20 days | 2.1739646428152604 | 1.6948058837225597 |
| 50 days | 2.175425417873673 | 1.6964519239787579 |

Table 4: Error Metrics of Exponential Smoothing

##### 5.1.3 Autoregressive Integrated Moving Average (ARIMA)

TODO MAKE WORK

### 5.2 Machine Learning Techniques Results

##### 5.2.1 Random Forest

Table 4 shows us the scores of the random forest models on the testing and more recent data with different numbers of days lag given. The model that performed the best against the test data had 40 days of lag and the following hyper parameters:

[Number of trees: 200, Max depth of the trees: 10, Minimum samples in a split: 5, Minimum samples in a leaf: 1]

The model that performed the best against the extrapolated data had 21 days of lag. Figure 6 shows its prediction against the actual difference values. This model had the following hyperparameters:

[Number of trees: 200, Max depth of the trees: 10, Minimum samples in a split: 10, Minimum samples in a leaf: 1]

|  |  |  |
| --- | --- | --- |
| **Random forest** | **RMSE** | **MAE** |
| **Days Lagged** | **1** | |
| Testing | 2.8327223045174983 | 1.5875324910530189 |
| Extrapolated | 3.6539386190256367 | 2.920929834626578 |
| **Days Lagged** | **3** | |
| Testing | 2.8121302513439175 | 1.587121395745055 |
| Extrapolated | 3.5843529991932743 | 2.8298826235148846 |
| **Days Lagged** | **5** | |
| Testing | 2.8330180316714966 | 1.5876364772819318 |
| Extrapolated | 3.53944606768763 | 2.776543711920735 |
| **Days Lagged** | **8** | |
| Testing | 2.8320632264684744 | 1.585865580303427 |
| Extrapolated | 3.5650494134225763 | 2.8061558285987296 |
| **Days Lagged** | **13** | |
| Testing | 2.8407717334953513 | 1.5892165703677603 |
| Extrapolated | 3.458721967354981 | 2.702624749138446 |
| **Days Lagged** | **21** | |
| Testing | 2.8031883388091243 | 1.5851285392663426 |
| Extrapolated | 3.4272562099946273 | 2.670331061855792 |
| **Days Lagged** | **40** | |
| Testing | 2.7943477544255626 | 1.5755179343110044 |
| Extrapolated | 3.5826487470491317 | 2.7506855587436525 |

Table : random forest result

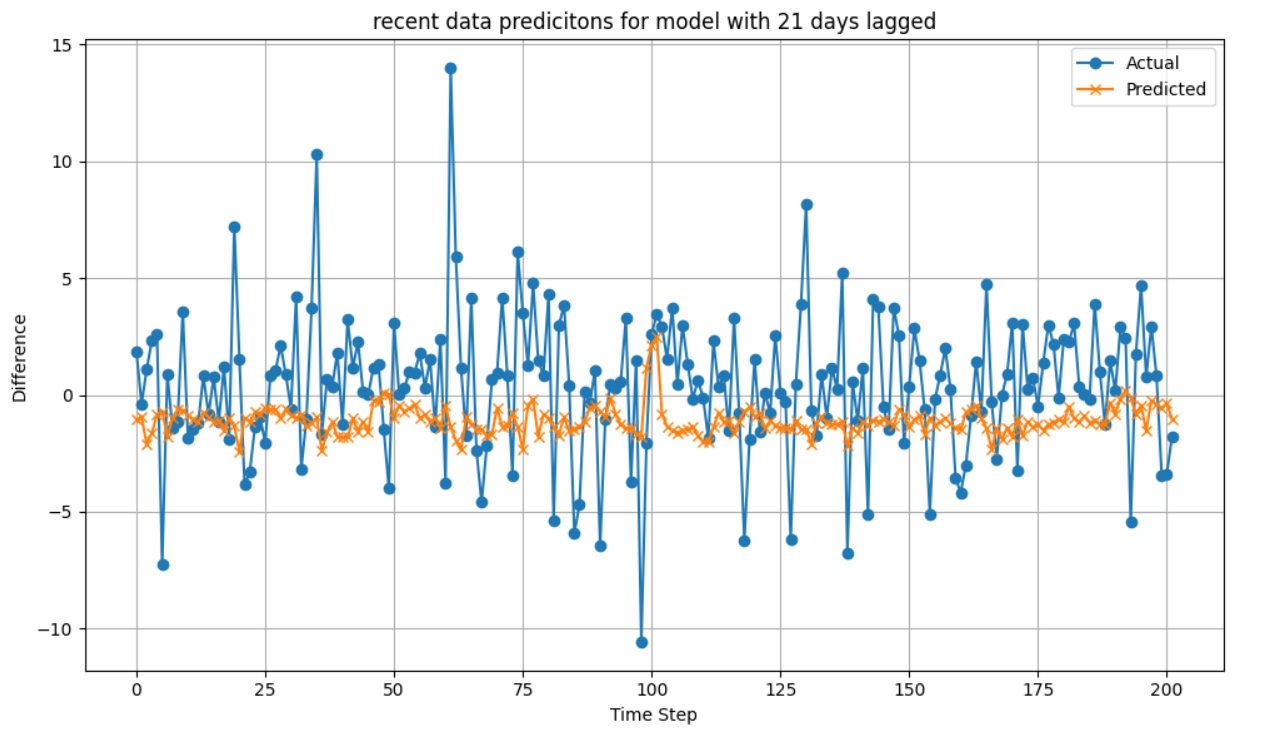


Figure prediction and actual difference values for random forest

##### 5.2.2 Gradient Boosting

Table 5 shows us the scores of the gradient boosting models on the testing and extrapolated data with different numbers of days lag given. The model that performed the best against the test data had 3 days of lag and the following hyper parameters:

[ l2\_regularization: 0.24324445943063866, learning rate: 0.051778878126471486, maximum bins: 137, maximum depth: 3, maximum iterations: 141, minimum samples in a lead: 23, soring: negative root mean squared error]

The model that performed the best against the extrapolated data had 20 days of lag. Figure 7 shows its prediction against the actual difference values. This model had the following hyperparameters:

[ l2\_regularization: 0.6040762215611126, learning rate: 0.013738386045000009, maximum bins: 230, maximum depth: 3, maximum iterations: 125, minimum samples in a lead: 45, soring: loss]

|  |  |  |
| --- | --- | --- |
| **Gradient Boosting** | **RMSE** | **MAE** |
| **Days lagged** | **1** | |
| Testing | 1.7236815046379101 | 0.969372496950562 |
| Extrapolated | 3.349612103572246 | 2.585726458632901 |
| **Days lagged** | **3** | |
| Testing | 1.7038002842977344 | 0.9642801126073656 |
| Extrapolated | 3.454720362291837 | 2.630580578511071 |
| **Days lagged** | **5** | |
| Testing | 1.7657769835808836 | 0.9871740711352227 |
| Extrapolated | 3.306656174541942 | 2.5616666370548984 |
| **Days Lagged** | **8** | |
| Testing | 1.7657769835808836 | 0.9871740711352227 |
| Extrapolated | 3.306656174541942 | 2.5616666370548984 |
| **Days Lagged** | **13** | |
| Testing | 1.7160160947190646 | 0.9706378845361063 |
| Extrapolated | 3.3317622353551006 | 2.5227883319434117 |
| **Days Lagged** | **20** | |
| Testing | 1.7193501460766578 | 0.9762195805082002 |
| Extrapolated | 3.1569929532466263 | 2.388517247990997 |
| **Days Lagged** | **40** | |
| Testing | 1.725027220196372 | 0.9811215403350901 |
| Extrapolated | 3.2158127267884216 | 2.433303911011357 |

Table Gradient boosting Results

A graph with blue and orange lines

AI-generated content may be incorrect.

Figure : Prediction and actual difference values for gradient boosting

##### 5.2.4 K-Nearest-Neighbour (KNN)

Table 6 shows us the scores of the KL Nearest Neighbour on the testing and extrapolated data with different numbers of days lag given. The model that performed the best against the test data had 5 days of lag and the following hyper parameters:

[Distance Metric: Euclidean, Number of Neighbours: 9, Weights: Uniform]

The model that performed the best against the extrapolated data had 20 days of lag. Figure 8 shows its prediction against the actual difference values. This model had the following hyperparameters:

[Distance Metric: Manhattan, Number of Neighbours: 9, Weights: Uniform]

|  |  |  |
| --- | --- | --- |
| **KNN** | **RMSE** | **MAE** |
| **Days lagged** | **1** | |
| Testing | 1.8012078407417198 | 1.0349009636047486 |
| Extrapolated | 3.225147987199131 | 2.3844728123630237 |
| **Days lagged** | **3** | |
| Testing | 1.7757879888510921 | 1.0130358563290465 |
| Extrapolated | 3.242198455545076 | 2.4434106824683934 |
| **Days lagged** | **5** | |
| Testing | 1.741921101394537 | 0.9909431659900867 |
| Extrapolated | 3.240361310938407 | 2.4546811614755226 |
| **Days Lagged** | **8** | |
| Testing | 1.7942922431424253 | 0.9889980126190948 |
| Extrapolated | 3.275663709113859 | 2.5125316036547503 |
| **Days Lagged** | **13** | |
| Testing | 1.7420580466024105 | 0.9750754439436996 |
| Extrapolated | 3.163703854664226 | 2.3633897485512714 |
| **Days Lagged** | **20** | |
| Testing | 1.7686118376190838 | 0.9945944169381478 |
| Extrapolated | 3.2029554174736226 | 2.430169771737916 |
| **Days Lagged** | **40** | |
| Testing | 1.7652996110021713 | 0.9937300118836794 |
| Extrapolated | 3.2170838136476294 | 2.4466679513257725 |

Table :KNN Results

A graph with blue and orange lines

AI-generated content may be incorrect.

Figure : Prediction and actual difference values for K-Nearest-Neighbour

5.2.5 Support Vector Machines (SVM)

Table 7 shows us the scores of the Support Vector Machine on the testing and extrapolated data with different numbers of days lag given. The model that performed the best against the test data had 5 days of lag and the following hyper parameters:

[C: 0.1, Degree: 4, epsilon: 0.01, gamma: auto, kernel: rbf]

The model that performed the best against the extrapolated data had 20 days of lag. Figure 8 shows its prediction against the actual difference values. This model had the following hyperparameters:

[C: 0.1, Degree: 4, epsilon: 0.1, gamma: auto, kernel: poly]

|  |  |  |
| --- | --- | --- |
| **SVM** | **RMSE** | **MAE** |
| **Days lagged** | **1** | |
| Testing | 1.721273666047767 | 0.9614052286698227 |
| Extrapolated | 3.255124192727642 | 2.5235905483844836 |
| **Days lagged** | **3** | |
| Testing | 1.718994626309663 | 0.9595984454734692 |
| Extrapolated | 3.252048706898524 | 2.526026360048861 |
| **Days lagged** | **5** | |
| Testing | 1.7188720766431804 | 0.9575146804850492 |
| Extrapolated | 3.2331808567684566 | 2.506745771782248 |
| **Days Lagged** | **8** | |
| Testing | 1.7204643423502999 | 0.9571972078585367 |
| Extrapolated | 3.2661357619525377 | 2.5484655926034656 |
| **Days Lagged** | **13** | |
| Testing | 1.729325634706246 | 0.961692705912958 |
| Extrapolated | 3.1792168231280327 | 2.4486809940266725 |
| **Days Lagged** | **20** | |
| Testing | 1.7278861688406773 | 0.9605725220215898 |
| Extrapolated | 3.0835853331140237 | 2.3167512370565158 |
| **Days Lagged** | **40** | |
| Testing | 1.7391313480676776 | 0.9674918727978197 |
| Extrapolated | 3.305295097898112 | 2.486142662437748 |

A graph with blue dots and a line

AI-generated content may be incorrect.

Figure : Prediction and actual difference values for K-Nearest-Neighbour

### 5.4 Artificial Neural Networks

##### 5.3.1 Single Hidden layer model

Table 8 shows us the scores of the Single layer ANN on the testing and extrapolated data with different numbers of days lag given. The model that performed the best against the test data had 8 days of lag and the following hyper parameters:

[Hidden Layer Size: 256, Activation Function: ReLU, Learning Rate: 0.01]

The model that performed the best against the extrapolated data had 20 days of lag. Figure 9 shows its prediction against the actual difference values. This model had the following hyperparameters:

[Hidden Layer Size: 224, Activation Function: sigmoid, Learning Rate: 0.0001]

|  |  |  |
| --- | --- | --- |
| **Single Layer ANN** | **RMSE** | **MAE** |
| **Days lagged** | **1** | |
| Testing | 1.7282888742812637 | 0.9539532609947806 |
| Extrapolated | 3.13454307645576 | 2.3710975646972656 |
| **Days lagged** | **3** | |
| Testing | 1.7418443264781953 | 0.9674590875287535 |
| Extrapolated | 3.023970367235993 | 2.1945786476135254 |
| **Days lagged** | **5** | |
| Testing | 1.7150205778067542 | 0.9590504017747449 |
| Extrapolated | 3.2482945295599417 | 2.506436586380005 |
| **Days Lagged** | **8** | |
| Testing | 1.6981226599908819 | 0.9534604050357359 |
| Extrapolated | 3.216242285555006 | 2.4690022468566895 |
| **Days Lagged** | **13** | |
| Testing | 1.7153736321420723 | 0.9477878768650683 |
| Extrapolated | 3.3672466936683607 | 2.627452850341797 |
| **Days Lagged** | **20** | |
| Testing | 1.7172931429310183 | 0.965040498836716 |
| Extrapolated | 3.272310559156763 | 2.493133068084717 |
| **Days Lagged** | **40** | |
| Testing | 1.724845536805612 | 0.956036229719626 |
| Extrapolated | 3.0423932884140275 | 2.200075149536133 |

Table Single Layer ANN Results

A graph with blue and orange lines

AI-generated content may be incorrect.

Figure : Prediction and actual difference values for Single Layer ANN

##### 5.3.2 Multiple Hidden Layers ANNs

Table 9 shows us the scores of the ANNs with multiple hidden layers. The table gives the performance metrics for the testing and extrapolated data with the different number of days lag. The model that performed the best on the testing data had 3 hidden layers and 8 days of lag with the followi9ng hyperparameters:

[First Hidden Layer Size: 64, First Layer Activation Function: RelU,

Second Hidden Layer Size: 192, Second Layer Activation Function: Sigmoid

Third Hidden Layer Size: 96, Third Layer Activation Function: Sigmoid

Learning Rate: 0.0001]

The model that performed the best against the extrapolated data had 8 hidden layers and 13 days of lag. Figure 9 shows its prediction against the actual difference values. This model had the following hyperparameters:

[First Hidden Layer Size: 64, First Layer Activation Function: RelU,

Second Hidden Layer Size: 192, Second Layer Activation Function: Sigmoid

Third Hidden Layer Size: 96, Third Layer Activation Function: Sigmoid

Learning Rate: 0.0001]

{'units\_0': 160, 'activation\_0': 'relu', 'units\_1': 64, 'activation\_1': 'sigmoid', 'units\_2': 192, 'activation\_2': 'relu', 'units\_3': 96, 'activation\_3': 'sigmoid', 'units\_4': 128, 'activation\_4': 'sigmoid', 'units\_5': 96, 'activation\_5': 'tanh', 'units\_6': 192, 'activation\_6': 'relu', 'units\_7': 160, 'activation\_7': 'tanh', 'learning\_rate': 0.001}

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Multiple Hidden  Layers ANNs | **2 Layers** | | **3 Layers** | | **5 Layers** | | **8Layers** | |
| Metric | RMSE | MAE | RMSE | MAE | RMSE | MAE | RMSE | MAE |
| **Days lagged** | **1** | | | | | | | |
| Testing | 1.7062 | 0.9583 | 1.7058 | 0.9551 | 1.7103 | 0.9563 | 1.7212 | 0.9599 |
| Extrapolated | 3.0617 | 2.2256 | 3.1472 | 2.3402 | 3.2050 | 2.4532 | 3.3130 | 2.5947 |
| **Days lagged** | **3** | | | | | | | |
| Testing | 1.70385 | 0.9435 | 1.7135 | 0.9538 | 1.7137 | 0.9541 | 1.7287 | 0.9614 |
| Extrapolated | 3.06172 | 2.2256 | 3.0300 | 2.2229 | 3.1363 | 2.3929 | 3.0274 | 2.2139 |
| **Days lagged** | **5** | | | | | | | |
| Testing | 1.7234 | 0.9514 | 1.7275 | 0.9552 | 1.7145 | 0.9510 | 1.7169 | 0.9555 |
| Extrapolated | 3.1027 | 2.3424 | 3.1155 | 2.3722 | 3.1760 | 2.4433 | 3.1968 | 2.4721 |
| **Days Lagged** | **8** | | | | | | | |
| Testing | 1.7296 | 0.9546 | 1.6885 | 0.9506 | 1.7194 | 0.9571 | 1.7308 | 0.9602 |
| Extrapolated | 3.1100 | 2.3404 | 3.0390 | 2.2519 | 3.2360 | 2.5088 | 3.0675 | 2.2863 |
| **Days Lagged** | **13** | | | | | | | |
| Testing | 1.7071 | 0.9463 | 1.7211 | 0.9527 | 1.7155 | 0.9520 | 1.7075 | 0.9500 |
| Extrapolated | 3.2117 | 2.4240 | 3.2202 | 2.4691 | 3.3442 | 2.6029 | 3.02201 | 2.2180 |
| **Days Lagged** | **20** | | | | | | | |
| Testing | 1.7267 | 0.9573 | 1.7313 | 0.9591 | 1.7236 | 0.9605 | 1.7296 | 0.9621 |
| Extrapolated | 3.0775 | 2.2853 | 3.0581 | 2.2703 | 3.0274 | 2.2135 | 3.0297 | 2.2182 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Multiple Hidden  Layers ANNs | **13 Layers** | | **20 Layers** | |
| Metric | RMSE | MAE | RMSE | MAE |
| **Days lagged** | **1** | | | |
| Testing | 1.7297 | 0.9622 | 1.7299 | 0.9622 |
| Extrapolated | 3.02981 | 2.2184 | 3.0301 | 2.2189 |
| **Days lagged** | **3** | | | |
| Testing | 1.7298 | 0.9621 | 1.7297 | 0.96212 |
| Extrapolated | 3.0299 | 2.2185 | 3.0298 | 2.21839 |
| **Days lagged** | **5** | | | |
| Testing | 1.7388 | 0.9582 | 1.7297 | 0.9622 |
| Extrapolated | 3.0925 | 2.3378 | 3.0298 | 2.2183 |
| **Days Lagged** | **8** | | | |
| Testing | 1.7199 | 0.9567 | 1.7296 | 0.9621 |
| Extrapolated | 3.0209 | 2.2063 | 3.0297 | 2.2182 |
| **Days Lagged** | **13** | | | |
| Testing | 1.7297 | 0.9622 | 1.7297 | 0.9621 |
| Extrapolated | 3.0299 | 2.2185 | 3.0299 | 2.2185 |
| **Days Lagged** | **20** | | | |
| Testing | 1.7296 | 0.9622 | 1.7295 | 0.9622 |
| Extrapolated | 3.0297 | 2.2181 | 3.0296 | 2.2179 |

Table : Multiple Hidden Layer ANNs Results

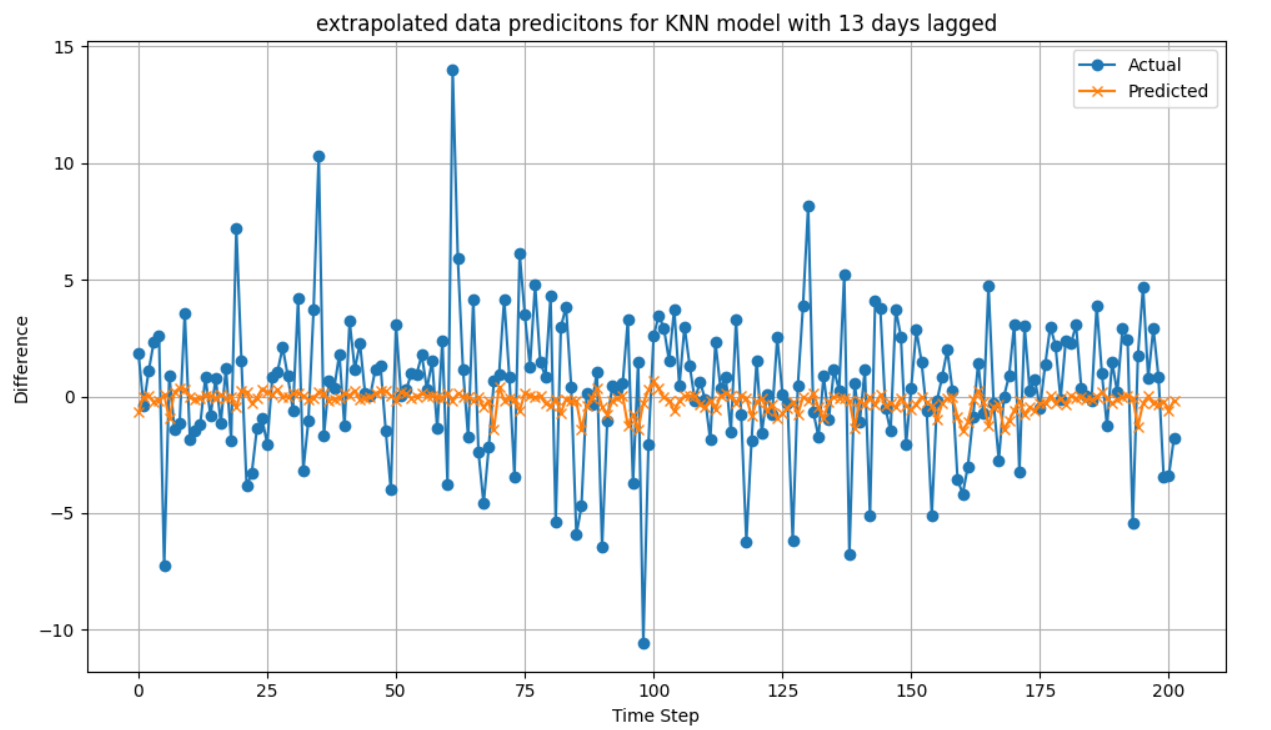


Figure : Prediction and actual difference values for best Multiple Layer ANN

##### 5.3.3 Recurrent Neural Networks and Long-short term memory

### 5.5 Further predictions

* 1. Predicting multiple days
     1. Targeting single X Day in the future
     2. Targeting all days up to X days in the future
        1. Single prediction
        2. Recursive

## **Comparison and discussion**

1. Single Next day prediction
   1. Compare models using root mean squared error, mean absolute error, and other metrics.
   2. Compare correct direction prediction.
   3. Determine percentage return if invested using model.
   4. Graph actual vs target, error and return on investment.
2. Multiple prediction days
   1. Compare models’ prediction score using average root mean squared error of days.
   2. Compare average correctly predicted days for each prediction.
   3. Determine percentage return on investment.
   4. Graph actual vs target, error and return on investment.

## **Discussion**

Comparison with metrics found in literature review.

## **Conclusion**

## **References**

Adebiyi, A. A., Adewumi, A. O. and Ayo, C. K. (2014) ‘Comparison of ARIMA and Artificial Neural Networks models for stock price prediction’, *Journal of Applied Mathematics*, 2014, Article 614342.

* 1. Ariyo, A. O. Adewumi and C. K. Ayo, “Stock Price Prediction Using the ARIMA Model,” *2014 UKSim-AMSS 16th International Conference on Computer Modelling and Simulation*, Cambridge, UK, 2014, pp. 106-112, doi: 10.1109/UKSim.2014.67.

Almasarweh, M. and Al Wadi, S. (2018) ‘ARIMA model in predicting banking stock market data’, *Published by Canadian Center of Science and Education*, pp. 309-312

Ball, R. (1978) ‘Anomalies in relationships between securities’ yields and yield-surrogates’, *Journal of Financial Economics*, 6, pp. 1033-1126. North-Holland Publishing Company.

Fama, E. F. (1965) ‘The behaviour of stock market prices’, *Journal of Business*, 38(1), pp. 34-105.

Jensen, M. C. (1978) ‘Some anomalous evidence regarding market efficiency’, *Journal of Financial Economics*, 6, pp. 95-101.

Meesad, P. and Rasel, R. I. (2013) ‘Predicting stock market price using support vector regression’, *2013 International Conference on Informatics, Electronics and Vision (ICIEV)*, Dhaka, Bangladesh, pp. 1-6. Doi: 10.1109/ICIEV.2013.6572570.

Meyler, A., Kenny, G., and Quinn, T. (1998) *Forecasting Irish inflation using ARIMA models*, MPRA Paper No. 11359

Sadorsky, P. (2021) ‘A random forests approach to predicting clean energy stock prices’, *Journal of Risk and Financial Management*, 14(2), p. 48.

scikit-learn (n.d.) *sklearn.preprocessing.MinMaxScaler*. Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html> (Accessed: 8 April 2025).

**What Follows is not apart of the final report draft but a summary of progress, problems, changes in scope, possible improvement, future plans and   
  
Description of Progress**

So far, I have created several juypter notebooks that do the following:

1. Import historical stock data from yahoo finance, so far, I have limited myself to apple stock from the past 10 years.
2. Preprocess the data, cleaning missing values and scale the data to remove the effect of larger numbers.
3. Create indicators like moving averages for each day.
4. Select Target for each prediction.
5. Sperate data in training and test data
6. Create models according to specific techniques with different hyperparameters.
7. Train models on training data
8. Test models on test data.
9. Compare error metrics of models.
10. Display graph of best model. Compare predicted vs actual value.
11. Select the best performing model and use it to predict data outside of training and test data scope.
12. Display graph and metrics of results.
13. Display percentage of correctly predicted directions

So far, I have done this for the following types of models:

* Random forest
* Support vector machines.
* Gradient boosting
* Single layer artificial neural networks
* Multiple layer artificial neural networks

I have also created a single layer artificial neural network to predict x number of days in the future and am looking to expand on this more.

The Code can be found in my git hub repo: https://github.com/KERPalmer/university\_final\_year\_project

**Problems and changes in direction**

Early in the project I created some notebooks to graph some ARIMA models and simple moving averages, while not entirely useless I will need to figure out how to include them, perhaps pivoting to using them as comparison as well as the literature view material.

At the beginning of this project, I mentioned the idea of creating an app for this project but that was quickly realized as being out of scope and unnecessary.

I had also expressed an interest in using semantic analysis with scraped social media data. If I have time to explore this option, I would like to research it, but I believe I will struggle to get a working model. I must also consider using that time to further improve existing models or expand in other directions.

One major problem I encountered was using different operating systems. Until this year I have always developed using a Mac and Mac OS. Recently I built a windows computer and am using it for the bulk of building and training prediction models. Installing and running these two different operating systems, one of which I was unfamiliar with, proved to be a challenge.

I encountered several issues with installing a version of python compatible with the python version I was using, this significantly ate up into the time I had assigned to the project and did cause a lot of frustration.

One of the major changes in direction that was inspired by my supervisor was to expand the prediction window. I was going to focus on building single day prediction models but have now decided to pivot to include the following ideas.

* Predicting a day further in the future, for example two days after
* Predicting multiple days at once, for example the next 3 days
* Using previous predictions to predict the next day, recursively.

I am hoping that this will make up the bulk of the project and can use any excess time to improve the models.

**Revised Work plan**

The current work plan is as follows.

Starting from the beginning of February and semester 2:

* Week 1: research gradient boosting and find research materials.
* Week 2-3: expand prediction of multiple days to other model types.
* Week 3-4: use model prediction to create return on investment metric.
* Week 5: compares model performances.
* Week 6: considers and implement improvements.
* Week 7- 9: finalise write up